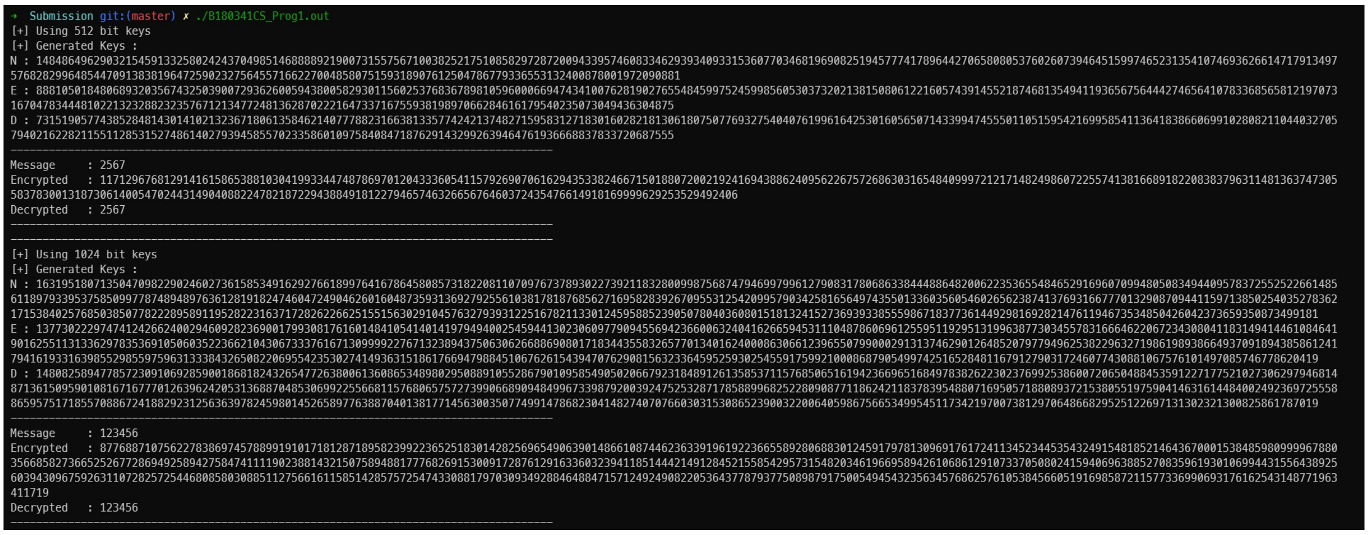
Q1 : Implementation of RSA using 512 bit and 1024 bit parameters

```
// Implementation of RSA algorithm using 512 Bit and 1024 Bit keys

#include <NTL/ZZ.h>

using namespace std;
using namespace NTL;

#define ERR_THRESHOLD 1000
// Error in range 2^(-ERR_THRESHOLD)

Vec<ZZ> KeyGen(long BIT_LENGTH)
{
   ZZ p , q , n , phi , e , d ;
   Vec<ZZ> v ;
   v.FixLength(3) ;

   GenPrime(p,BIT_LENGTH,ERR_THRESHOLD) ;
   GenPrime(q,BIT_LENGTH,ERR_THRESHOLD) ;
   n = p * q ;
   phi = (p-1) * (q-1) ;

   ZZ t ;
   t = 0 ;
   e = 0 ;
   while (t != 1 && e != 1)
   {
      e = RandomBnd(phi) ;
      t = GCD(e,phi) ;
   }

   d = InvMod(e,phi) ;

   v[0] = n ;
   v[1] = e ;
   v[2] = d ;
   return v ;
}

// Encrypt
ZZ encrypt(ZZ m,ZZ e,ZZ n)
{
   ZZ c ;
   c = PowerMod(m,e,n) ;
   return c ;
}

// Decrypt
ZZ decrypt(ZZ c,ZZ d,ZZ n)
{
   ZZ m ;
```

```
    m = PowerMod(c,d,n) ;
    return m ;
}


int main()
{
    ZZ p , q , n , phi , e , d ;

    ZZ m , c , t ;

    long KEY_SIZE = 512 ;
    cout << "[+] Using " << KEY_SIZE << " bit keys" << endl ;

    Vec<ZZ> keys = KeyGen(KEY_SIZE) ;
    n = keys[0] ;
    e = keys[1] ;
    d = keys[2] ;

    cout << "[+] Generated Keys : " << endl ;
    cout << "N : " << n << endl ;
    cout << "E : " << e << endl ;
    cout << "D : " << d << endl ;
    cout << "-----------------------------------------------------------------------
---------------" << endl ;

    m = 2567 ;
    cout << "Message     : " << m << endl ;
    c = encrypt(m,e,n) ;
    cout << "Encrypted   : " << c << endl ;
    t = decrypt(c,d,n) ;
    cout << "Decrypted   : " << t << endl ;
    cout << "-----------------------------------------------------------------------
---------------" << endl ;
    cout << "-----------------------------------------------------------------------
---------------" << endl ;

    KEY_SIZE = 1024 ;
    cout << "[+] Using " << KEY_SIZE << " bit keys" << endl ;

    keys = KeyGen(KEY_SIZE) ;
    n = keys[0] ;
    e = keys[1] ;
    d = keys[2] ;

    cout << "[+] Generated Keys : " << endl ;
    cout << "N : " << n << endl ;
    cout << "E : " << e << endl ;
    cout << "D : " << d << endl ;
    cout << "-----------------------------------------------------------------------
---------------" << endl ;

    m = 123456 ;
    cout << "Message     : " << m << endl ;
    c = encrypt(m,e,n) ;
    cout << "Encrypted   : " << c << endl ;
    t = decrypt(c,d,n) ;
    cout << "Decrypted   : " << t << endl ;
    cout << "-------------------------------------------------------------------------------
---------------" << endl ;
}
```

Output :



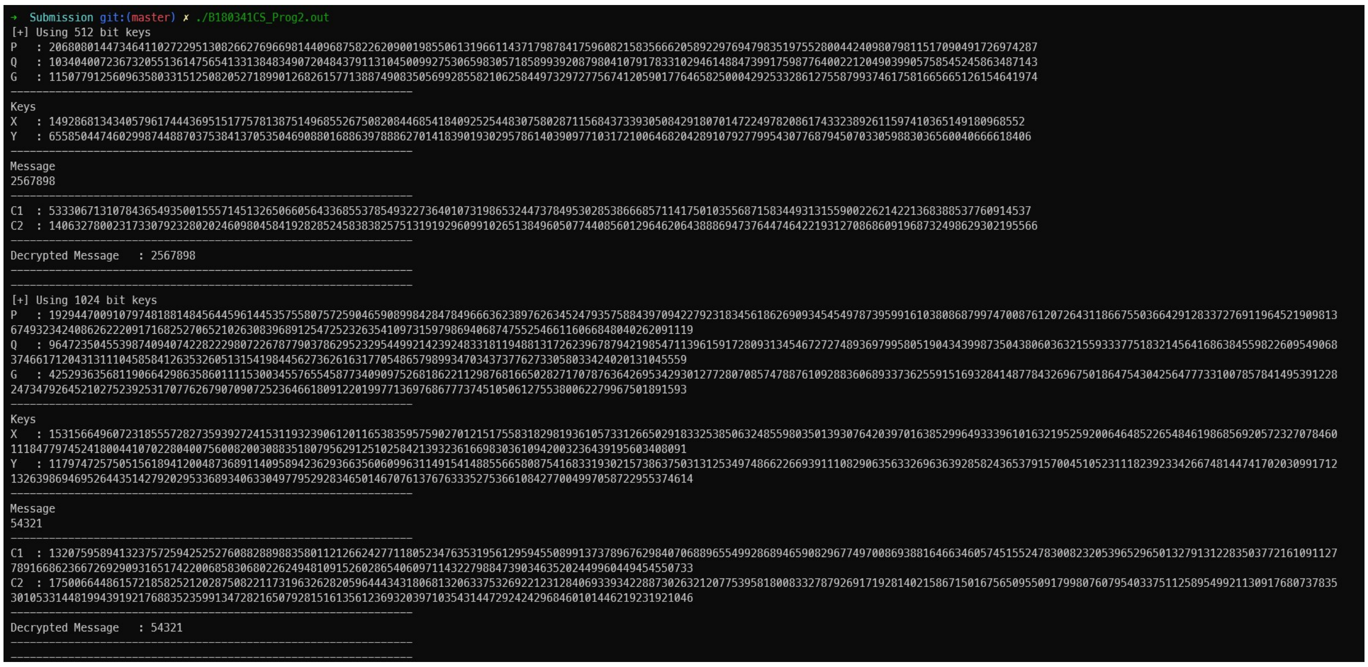
```
→ Submission git:(master) ✗ ./B180341CS_Prog1.out
[+] Using 512 bit keys
[+] Generated Keys :
N : 148486496290321545913325802424370498514688889219007315575671003825217510858297287200943395746083346293934093315360770346819690825194577741789644270658080537602607394645159974652313541074693626614717913497
576828299648544709138381964725902327564557166227004858075159318907612504786779336553132400878001972090881
E : 888105018480689320356743250390072936260059438005829301156025376836789810596000669474341007628190276554845997524599856053037320213815080612216057439145521874681354941193656756444274656410783368565812197073
16704783444810221323288232357671213477248136287022216473371675593819897066284616179540235073049436304875
D : 731519057743852848143014102132367180613584621407778823166381335774242137482715958312718301602821813061807507769327540407619961642530160565071433994745550110515954216995854113641838660699102808211044032705
79402162282115511285315274861402793945855702335860109758408471876291432992639464761936668837833720687555
--------------------------------------------------------------------------------------
Message      : 2567
Encrypted    : 117129676812914161586538810304199334474878697012043336054115792690706162943533824667150188072002192416943886240956226757268630316548409997212171482498607225574138166891822083837963114813637473 05
58378300131873061400547024431490408822478218722943884918122794657463266567646037243547661491816999962925352949 2406
Decrypted    : 2567
--------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------
[+] Using 1024 bit keys
[+] Generated Keys :
N : 163195180713504709822902460273615853491629276618997641678645808573182208110709767378930227392118328009987568747946997996127908317806863384448864820062235365548465291696070994805083494409578372552522661485
611897933953758509977874894897636128191824746047249046260160487359313692792556103817818768562716958283926709553125420995790342581656497435501336035605460265623874137693166777013290870944115971385025403 5278362
171538402576850385077822289589119528223163717282622662515515630291045763279393122516782113301245958852390507804036080151813241527369393385559867183773614492981692821476119467353485042604237365935087349918 1
E : 137730222974741242662400294609282369001799308176160148410541401419794940025459441302306097790945569423660063240416266594531110487860696125595119295131996387730345578316664622067234308041183149414461084641
901625511313362978353691050603522366210430673337616713099992276713238943750630626688690801718344355832657701340162400086306612396550799000291313746290126485207977949625382296327198619893866493709189438586124 1
794161933163985529855975963133384326508220695542353027414936315186176694798845106762615439470762908156323364595259302545591759921000868790549974251652848116791279031724607743088106757610149708574677 8620419
D : 148082589477857230910692859001868182432654772638006136086534898029508891055286790109585490502066792318489126135853711576850651619423669651684978382622302376992538600720650488453591227177521027306297946814
871361509590108167167770126396242053136887048530699225566811576806575727399066890948499673398792003924752532871785889968252280908771186242118378395488071695057188089372153805519759041463161448400249236972555 8
865957517185570886724188292312563639782459801452658977638870401381771456300350774991478682304148274070766030315308652390032200640598675665349954511734219700738129706486682952512269713130232130082586178701 9
--------------------------------------------------------------------------------------
Message      : 123456
Encrypted    : 877688710756227838697457889919101718128718958239922365251830142825696549063901486610874462363391961922366558928068830124591797813096917617241134523445354324915481852146436700015384859809999 67880
356685827366525267728694925894275847411119023881432150758948817776826915300917287612916336032394118514442149128452155854295731548203461966958942610686129107337050802415940696388527083596193010699443155643892 5
603943096759263110728257254468085803088511275661611585142857572547433088179703093492884648847157124924908220536437787937750898791750054945432356345768625761053845660519169858721157733699069317616254314877196 3
411719
Decrypted    : 123456
--------------------------------------------------------------------------------------
```

Q2 : Implementation of ElGamal using 512 bit and 1024 bit parameters

```
// Implementation of ElGamal algorithm using 512 Bit and 1o24 Bit keys

#include <NTL/ZZ.h>

using namespace std;
using namespace NTL;

#define ERR_THRESHOLD 1000
// Error in range 2^(-ERR_THRESHOLD)

Vec<ZZ> Setup(long BIT_LENGTH)
{
    Vec<ZZ> v ;
    v.FixLength(3) ;

    ZZ p , q , t , g , h;

    GenGermainPrime(q,BIT_LENGTH,ERR_THRESHOLD) ;
    p = 2*q + 1 ;

    g = 1 ;
    while(g==1)
    {
        h = RandomBnd(p) ;
        g = PowerMod(h,(p-1)/q,p) ;
    }

    v[0] = p ;
    v[1] = q ;
    v[2] = g ;
    return v ;
}

// Key Generation
Vec<ZZ> KeyGen(ZZ p,ZZ q,ZZ g)
{
    Vec<ZZ> v ;
    v.FixLength(2) ;

    ZZ x,y ;

    do {
        x = RandomBnd(q) ;
    } while (GCD(x,p)!=1) ;

    y = PowerMod(g,x,p);
    v[0] = x ;
    v[1] = y ;

    return v ;
}

// Encryption
Vec<ZZ> encrypt(ZZ m,ZZ p,ZZ q,ZZ g,ZZ y)
{
    Vec<ZZ> v ;
    v.FixLength(2) ;

    ZZ k ;

    do {
        k = RandomBnd(q) ;
    } while (GCD(k,p)!=1) ;

    ZZ c1,c2 ;

    c1 = PowerMod(g,k,p) ;
    c2 = MulMod(m,PowerMod(y,k,p),p) ;

    v[0] = c1 ;
    v[1] = c2 ;
    return v;
}

// Decryption
ZZ decrypt(ZZ c1,ZZ c2,ZZ x,ZZ p)
{
    ZZ m ;
    m = MulMod(c2,PowerMod(c1,-x,p),p);
    return m ;
}

int main()
{
    ZZ p , q , n , phi , g , d ;
    long BIT_LENGTH ;

    BIT_LENGTH = 512 ;
    cout << "[+] Using " << BIT_LENGTH << " bit keys" << endl ;

    Vec<ZZ> set = Setup(BIT_LENGTH) ;
    p = set[0] ;
    q = set[1] ;
    g = set[2] ;

    cout << "P   : " << p << endl ;
    cout << "Q   : " << q << endl ;
    cout << "G   : " << g << endl ;
    cout << "----------------------------------------------------------------" <<
endl ;

    ZZ x,y ;
    Vec<ZZ> keys = KeyGen(p,q,g) ;
    x = keys[0] ;
    y = keys[1] ;

    cout << "Keys" << endl ;
    cout << "X   : " << x << endl ;
    cout << "Y   : " << y << endl ;
    cout << "----------------------------------------------------------------" <<
endl ;

    ZZ c1 , c2 ;
    ZZ m ;

    m = 2567898 ;
    cout << "Message" << endl ;
    cout << m << endl ;
    cout << "----------------------------------------------------------------" <<
endl ;
    Vec<ZZ> c = encrypt(m,p,q,g,y) ;
    c1 = c[0] ;
    c2 = c[1] ;

    cout << "C1  : " << c1 << endl ;
    cout << "C2  : " << c2 << endl ;
    cout << "----------------------------------------------------------------" <<
endl ;

    ZZ t ;
    t = decrypt(c1,c2,x,p) ;

    cout << "Decrypted Message   : " << t << endl ;
```

```
   cout << "---------------------------------------------------------------" <<
endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

   BIT_LENGTH = 1024 ;
   cout << "[+] Using " << BIT_LENGTH << " bit keys" << endl ;

   set = Setup(BIT_LENGTH) ;
   p = set[0] ;
   q = set[1] ;
   g = set[2] ;

   cout << "P   : " << p << endl ;
   cout << "Q   : " << q << endl ;
   cout << "G   : " << g << endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

   keys = KeyGen(p,q,g) ;
   x = keys[0] ;
   y = keys[1] ;

   cout << "Keys" << endl ;
   cout << "X   : " << x << endl ;
   cout << "Y   : " << y << endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

   m = 54321 ;
   cout << "Message" << endl ;
   cout << m << endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

   c = encrypt(m,p,q,g,y) ;
   c1 = c[0] ;
   c2 = c[1] ;

   cout << "C1  : " << c1 << endl ;
   cout << "C2  : " << c2 << endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

   t = decrypt(c1,c2,x,p) ;

   cout << "Decrypted Message   : " << t << endl ;
   cout << "---------------------------------------------------------------" <<
endl ;
   cout << "---------------------------------------------------------------" <<
endl ;

}
```

Output :



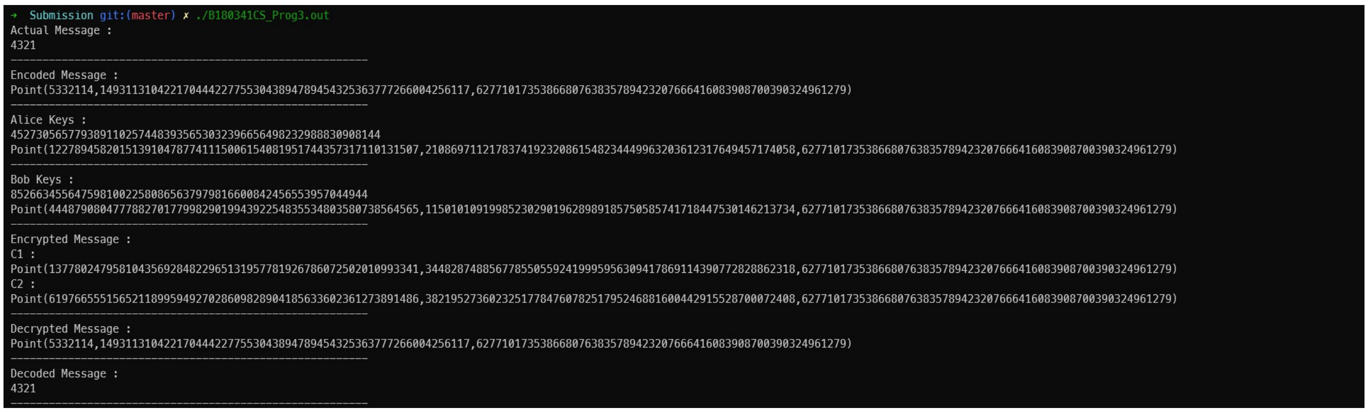
```
➜  Submission git:(master) ✗ ./B180341CS_Prog2.out
[+] Using 512 bit keys
P   : 20680801447346411027229513082662769669814409687582262090019855061319661143717987841759608215835666205892297694798351975528004424098079811517090491726974287
Q   : 10340400723673205513614756541331384834907204843791131045009927530659830571858993920879804107917833102946148847399175987764002212049039905758545245863487143
G   : 11507791256096358033151250820527189901268261577138874908350569928558210625844973297277567412059017764658250004292533286127558799374617581665665126154641974
----------------------------------------------------------------
Keys
X   : 149286813434057961744436951517757813875149685526750820844685418409252544830758028711568437339305084291807014722497820861743323892611597410365149180968552
Y   : 6558504474602998744887037538413705350469088016886397888627014183901930295786140390977103172100646820428910792779954307768794507033059883036560040666618406
----------------------------------------------------------------
Message
2567898
----------------------------------------------------------------
C1  : 5333067131078436549350015557145132650660564336855378549322736401073198653244737849530285386668571141750103556871583449313155900226214221368388537760914537
C2  : 14063278002317330792328020246098045841928285245838382575131919296099102651384960507744085601296462064388869473764474642219312708686091968732498629302195566
----------------------------------------------------------------
Decrypted Message   : 2567898
----------------------------------------------------------------
----------------------------------------------------------------
[+] Using 1024 bit keys
P   : 1929447009107974818814845644596144535755807572590465908998428478496663623897626345247935758843970942279231834561862690934545497873959916103808687997470087612072643118667550366429128337276911964521909813
67493234240862622209171682527065210263083968912547252326354109731597986940687475525466116066848040262091119
Q   : 9647235045539874094074228222980722678779037862952329544992142392483318119488131726239678794219854711396159172809313454672727489369799580519043439987350438060363215593337751832145641686384559822609549068
3746617120431311104585841263532605131541984456273626163177054865798993470343737762733058033424020131045559
G   : 4252936356811906642986358601111530034557655458773409097526818622112987681665028271707876364269534293012772807085747887610928836068933736255915169328414877843269675018647543042564777331007857841495391228
2473479264521027523925317077626790709072523646618091220199771369768677737451050612755380062279967501891593
----------------------------------------------------------------
Keys
X   : 1531566496072318555728273593927241531193239061201165383595759027012151755831829819361057331266502918332538506324855980350139307642039701638529964933396101632195259200646485226548461986856920572327078460
1118477974524180044107022804007560082003088351807956291251025842139323616698303610942003236439195603408091
Y   : 1179747257505156189412004873689114095894236293663560609963114915414885566580875416833193021573863750313125349748662266939111082906356332696363928582436537915700451052311182392334266748144741702030991712
13263986946952644351427920295336893406330497795292834650146707613767633352753661084277004997058722955374614
----------------------------------------------------------------
Message
54321
----------------------------------------------------------------
C1  : 1320759589413237572594252527608828898835801121266242771180523476353195612959455089913737896762984070688965549928689465908296774970086938816466346057451552478300823205396529650132791312283503772161091127
78916686236672692909316517422006858306802262494810915260286540609711432279884739034635202449960449454550733
C2  : 1750066448615721858252120287508221173196326282059644434318068132063375326922123128406933934228873026321207753958180083327879269171928140215867150167565095509179980760795403375112589549921130917680737835
30105331448199439192176883523599134728216507928151613561236932039710354314472924242968460101446219231921046
----------------------------------------------------------------
Decrypted Message   : 54321
----------------------------------------------------------------
----------------------------------------------------------------
```

Q3 : Implementation of ECC using F-192 parameters

```
// Implementation of ECC algorithm using 192 Bit Primary Field

#include <string.h>
#include <NTL/ZZ.h>
#include <vector>
#include <bitset>

#define K_VALUE 1234

using namespace NTL ;
using namespace std ;

ZZ modInverse(ZZ x,ZZ p)
{
    for(ZZ i=ZZ(0);i<p;i+=1)
        if(MulMod(i,x,p) == 1)
            return i ;
    return ZZ(-1) ;
}

class Point {
    private:
        ZZ p ;
        ZZ a ;
        ZZ b ;
        string zztostring(ZZ num)
        {
            long len = ceil(log(num)/log(128));
            char str[len];
            for(long i = len-1; i >= 0; i--)
            {
                str[i] = conv<int>(num % 128);
                num /= 128;
            }

            return (string) str;
        }
    public:
        ZZ x ;
        ZZ y ;

        Point(ZZ x_,ZZ y_,ZZ p_,ZZ a_=ZZ(0),ZZ b_=ZZ(0))
        {
            x = x_ ;
            y = y_ ;
            p = p_ ;
            a = a_ ;
            b = b_ ;
        }

        Point(const Point &P)
        {
            x = P.x ;
            y = P.y ;
            p = P.p ;
            a = P.a ;
            b = P.b ;
        }

        ZZ get_a()
        {
            return a ;
        }

        ZZ get_b()
        {
            return b ;
        }

        ZZ get_p()
        {
            return p ;
        }

        void display()
        {
            cout << "Point("
                 << x << ","
                 << y << ","
                 << p << ")" << endl ;
        }

        string toString()
        {
            string str = "Point(" ;
            str += zztostring(x) ;
            str += "," ;
            str += zztostring(y) ;
            str += "," ;
            str += zztostring(p) ;
            str += ")" ;
            return str ;
        }

        Point inverse()
        {
            Point P = copy() ;
            P.y = (-P.y) % P.p ;
            return P ;
        }

        Point copy()
        {
            Point P = Point(x,y,p,a,b) ;
            return P ;
        }

        bool Eq(Point const &obj)
        {
            if(obj.x == x && obj.y == y && obj.p == p && obj.a == a && obj.b == b)
                return 1 ;
            else
                return 0 ;
        }

        Point Double()
        {
            ZZ l ;
            ZZ t_y ;
            try {
                t_y = InvMod(MulMod(ZZ(2),y,p),p) ;
            } catch(InvModErrorObject &e) {
                cout << "Error: " << e.what() << endl ;
                return Point(ZZ(-1),ZZ(-1),ZZ(-1)) ;
            }
            catch(...) {
                cout << "[!] Double : Non Invertible Point" << endl ;
                return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
            }
            l = MulMod(ZZ(3) * power(x,2) + a,t_y,p) ;
            ZZ fx = (power(l,2)-x-x) % p ;
            ZZ t = x - fx ;
            ZZ fy = (l*t - y) % p ;

            return Point(fx,fy,p,a,b) ;
        }

        ostream& operator<<(ostream &out)
        {
            out << "Point(" << x << "," << y << "," << p << "," << a << "," << b
<< ")" ;
            return out ;
        }

        Point operator + (Point const &obj)
        {
            if(p != obj.p)
                cout << "[+] Invalid Operands" << endl ;
            ZZ y_ = obj.y - y ;
            ZZ x_ = obj.x - x ;

            if(obj.y == 0 && obj.x == 0)
                return Point(x,y,p,a,b) ;
            if(y == 0 && x == 0)
                return Point(obj.x,obj.y,obj.p,obj.a,obj.b) ;

            ZZ l ;
            if(Eq(obj))
            {
                ZZ t_y ;
                try {
                    t_y = InvMod(MulMod(ZZ(2),y,p),p) ;
                } catch(...) {
                    cout << "[!] + if : Non Invertible Point" << endl ;
                    return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
                }
                l = MulMod(ZZ(3) * power(x,2) + a,t_y,p) ;
            }
            else
            {
                ZZ t_x ;
                try {
                    x_ = x_ % p ;
                    t_x = InvMod(x_,p) ;
                } catch(...) {
                    cout << "[!] + else : Non Invertible Point : " << x_ << endl ;
                    cout << "[!] GCD : " << GCD(x_,p) << endl ;
                    return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
                }
                l = MulMod(y_,t_x,p) ;
            }
            ZZ fx = (power(l,2)-x-obj.x) % p ;
            ZZ t = x - fx ;
            ZZ fy = (l*t - y) % p ;

            return Point(fx,fy,p,a,b) ;
        }

        Point operator * (ZZ const &obj)
        {
            return scalarMul(obj) ;
        }
        Point operator * (int const &obj)
        {
            return scalarMul(ZZ(obj)) ;
        }

        // Scalar Multiplication of Point
        Point scalarMul(ZZ const &k)
        {
            Point P = copy() ;
            Point ans = Point(ZZ(0),ZZ(0),p,a,b) ;
            unsigned char* p = new unsigned char[NumBytes(k)];
            BytesFromZZ(p, k, NumBytes(k)); // pp = byte-representation of N
            for(int i=0;i<NumBytes(k);i+=1)
            {
                bitset<8> x(p[i]) ;         // x = binary representation of p[i]
                for(int j=0;j<8;j+=1)
                {
                    if(x[j] == 1)
                        ans = ans + P ;
                    P = P.Double() ;
                }
            }
            delete[] p;
            return ans ;
        }
} ;

typedef struct Key
{
    ZZ privateKey ;
    Point publicKey ;
} Key ;

Key KeyGen(ZZ n,Point B)
{
    ZZ PrivateKey = RandomBnd(n) ;
    Point PublicKey = B * PrivateKey ;
    Key k = {PrivateKey,PublicKey} ;
    return k ;
}

Point messageEncode(ZZ m,Point BasePoint)
{
    ZZ k = ZZ(K_VALUE) ;
    ZZ Xj , Yj , Sj ;

    for(int j=0;j<k;j+=1)
    {
        Xj = k * m + j ;
        Sj = power(Xj,3) + BasePoint.get_a()*Xj + BasePoint.get_b() ;
        Sj = Sj % BasePoint.get_p() ;
        if(Jacobi(Sj,BasePoint.get_p())==1)
        {
            Yj = PowerMod(Sj,(BasePoint.get_p()+1)/ZZ(4),BasePoint.get_p()) ;
            return
Point(Xj,Yj,BasePoint.get_p(),BasePoint.get_a(),BasePoint.get_b()) ;
        }
    }
    return BasePoint * m ;
}

ZZ messageDecode(Point Pm,Point BasePoint)
{
    ZZ k = ZZ(K_VALUE) ;
    return Pm.x / k ;
}

vector<Point> encrypt(ZZ k,Point BasePoint,Point Pm,Point publicKey)
{
    Point C1 = BasePoint * k ;
    Point C2 = Pm + (publicKey * k) ;

    vector<Point> cipher ;

    cipher.reserve(2) ;

    cipher.push_back(C1) ;
    cipher.push_back(C2) ;

    return cipher ;
}

Point decrypt(ZZ privateKey,Point C1,Point C2)
{
    Point t = (C1 * privateKey).inverse() ;
    Point Pm = C2 + t ;
    return Pm ;
}

int main()
{
    Vec<ZZ> curveParams ;
    curveParams.SetLength(3) ;
    curveParams[0] = power(ZZ(2),192) - power(ZZ(2),64) - ZZ(1) ;   // p
    curveParams[1] = ZZ(-3) ;                                       // a
    curveParams[2] = conv<ZZ>
("2455155546008943817740293915197451784769108058161191238065") ;    // b

    ZZ p = curveParams[0] ;
    ZZ a = curveParams[1] ;
    ZZ b = curveParams[2] ;

    ZZ n = conv<ZZ>("6277101735386680763835789423176059013767194773182842284081")
;    // n - Order
    ZZ seed = conv<ZZ>("275585503993527016686210752207080241786546919125") ;
    ZZ h = ZZ(1) ;
    ZZ r = conv<ZZ>("1191689908718309326471930603292001425137626342642504031845")
```

```
;

    ZZ x = conv<ZZ>("602046282375688656758213480587526111916698976636884684818") ;
    ZZ y = conv<ZZ>("174050332293622031404857552280219410364023488927386650641") ;

    Point BasePoint = Point(
        x,              // x
        y,              // y
        curveParams[0],
        curveParams[1],
        curveParams[2]
    ) ;

    Key key = KeyGen(n,BasePoint) ;

    int message = 4321 ;
    ZZ m = ZZ(message) ;

    Point Pm = messageEncode(m,BasePoint) ;

    cout << "Actual Message : " << endl ;
    cout << m << endl ;
    cout << "--------------------------------------------------------" << endl ;

    cout << "Encoded Message : " << endl ;
    Pm.display() ;
    cout << "--------------------------------------------------------" << endl ;

    Key aliceKey = KeyGen(n,BasePoint) ;
    cout << "Alice Keys : " << endl ;
    cout << aliceKey.privateKey << endl ;
    aliceKey.publicKey.display() ;
    cout << "--------------------------------------------------------" << endl ;

    Key bobKey = KeyGen(n,BasePoint) ;
    cout << "Bob Keys : " << endl ;
    cout << bobKey.privateKey << endl ;
    bobKey.publicKey.display() ;
    cout << "--------------------------------------------------------" << endl ;

    ZZ k = RandomBnd(curveParams[0]) ;

    vector<Point> cipher = encrypt(k,BasePoint,Pm,bobKey.publicKey) ;
    cout << "Encrypted Message : " << endl ;
    cout << "C1 : " << endl ;
    cipher[0].display() ;
    cout << "C2 : " << endl ;
    cipher[1].display() ;
    cout << "--------------------------------------------------------" << endl ;

    Point tm = decrypt(bobKey.privateKey,cipher[0],cipher[1]) ;
    cout << "Decrypted Message : " << endl ;
    tm.display() ;
    cout << "--------------------------------------------------------" << endl ;
    ZZ decodedMessage = messageDecode(tm,BasePoint) ;
    cout << "Decoded Message : " << endl ;
    cout << decodedMessage << endl ;
    cout << "---------------------------------------------------------" << endl ;
}
```

Output :



```
➜  Submission git:(master) ✗ ./B180341CS_Prog3.out
Actual Message :
4321
---------------------------------------------------------
Encoded Message :
Point(5332114,1493113104221704442277553043894789454325363777266004256117,6277101735386680763835789423207666416083908700390324961279)
---------------------------------------------------------
Alice Keys :
4527305657793891102574483935653032396656498232988830908144
Point(1227894582015139104787741115006154081951744357317110131507,2108697112178374192320861548234449963203612317649457174058,6277101735386680763835789423207666416083908700390324961279)
---------------------------------------------------------
Bob Keys :
85266345564759810022580865637979816600842456553957044944
Point(4448790804777882701779982901994392254835534803580738564565,1150101091998523029019628989185750585741718447530146213734,6277101735386680763835789423207666416083908700390324961279)
---------------------------------------------------------
Encrypted Message :
C1 :
Point(1377802479581043569284822965131957781926786072502010993341,3448287488567785505592419995956309417869114390772828862318,6277101735386680763835789423207666416083908700390324961279)
C2 :
Point(6197665551565211899594927028609828904185633602361273891486,3821952736023251778476078251795246881600442915528700072408,6277101735386680763835789423207666416083908700390324961279)
---------------------------------------------------------
Decrypted Message :
Point(5332114,1493113104221704442277553043894789454325363777266004256117,6277101735386680763835789423207666416083908700390324961279)
---------------------------------------------------------
Decoded Message :
4321
---------------------------------------------------------
```

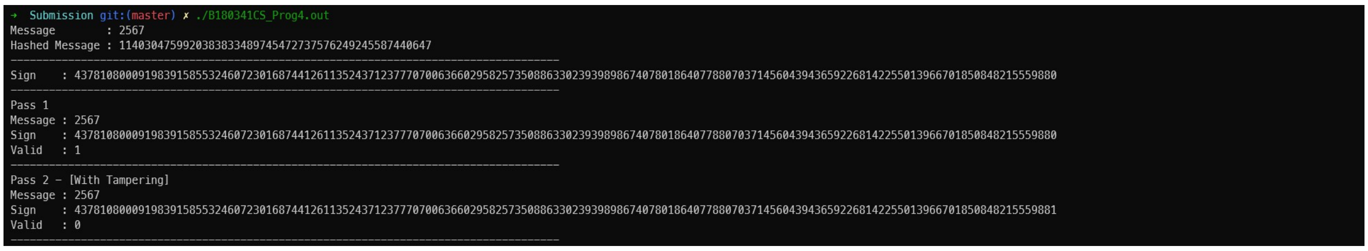
Q4 - a : Digital Signature Implementation using RSA

Header Files with Necessary Utils for SHA Hash

```
// Implementation of Digital Signature using RSA
// Hash related functions are stored in crypto.h

#include <NTL/ZZ.h>
#include <openssl/sha.h>
#include <bits/stdc++.h>
#include <string>

using namespace std;
using namespace NTL;

char *hexdigest(unsigned char *md, int len)
{
    static char buf[80];
    int i;
    for (i = 0; i < len; i++)
        sprintf(buf + i * 2, "%02x", md[i]);
    return buf;
}

ZZ hexToZZ(char *hex)
{
   ZZ res = ZZ(0);
   int i;
   for (i = 0; i < strlen(hex); i += 1)
   {
      res <<= 4;
      char x = hex[i];
```

```
        if(x>='0' && x <='9')
           res += hex[i]-48;
        else if(x>='a' && x <='f')
           res += hex[i]-87;
        else if(x>='A' && x <='F')
           res += hex[i]-55;
    }
    return res ;
}

string numberToString(ZZ num)
{
    string s = "";
    while (num > 0)
    {
        s += (num % 10) + '0';
        num /= 10;
    }
    reverse(s.begin(), s.end());
    return s;
}

// SHA1
ZZ Hash(string s)
{
    unsigned char hash[SHA_DIGEST_LENGTH]; // == 20

    SHA_CTX sha1;
    SHA1_Init(&sha1);
    SHA1_Update(&sha1, s.c_str(), s.length());
    SHA1_Final(hash, &sha1);

    ZZ h = hexToZZ(hexdigest(hash, SHA_DIGEST_LENGTH));
    return h ;
}

ZZ Hash(ZZ m)
{
    string s = numberToString(m);
    return Hash(s);
}

ZZ Hash(char *m)
{
    string s = m;
    return Hash(s);
}

#define BIT_LENGTH 512
#define ERR_THRESHOLD 1000
// Error in range 2^(-ERR_THRESHOLD)

Vec<ZZ> KeyGen()
{
   ZZ p , q , n , phi , e , d ;
   Vec<ZZ> v ;
   v.FixLength(3) ;

   GenPrime(p,BIT_LENGTH/2,ERR_THRESHOLD) ;
   GenPrime(q,BIT_LENGTH/2,ERR_THRESHOLD) ;
   n = p * q ;
   phi = (p-1) * (q-1) ;

   ZZ t ;
   t = 0 ;
   e = 0 ;
   while (t != 1 && e != 1)
   {
      e = RandomBnd(phi) ;
      t = GCD(e,phi) ;
   }
   d = InvMod(e,phi) ;

   v[0] = n ;
   v[1] = e ;
   v[2] = d ;
   return v ;
}

//Sign
ZZ Sign(ZZ m,ZZ d,ZZ n)
{
   ZZ c ;
   ZZ h = Hash(m) ;
   c = PowerMod(h,d,n) ;
   return c ;
}

// Verify
bool Verify(ZZ m,ZZ sigma,ZZ e,ZZ n)
{
   ZZ c ;
   ZZ h = Hash(m) ;
   ZZ h_ = PowerMod(sigma,e,n) ;
   if (h == h_)
      return 1 ;
   else
      return 0 ;
}

int main()
{
   ZZ p , q , n , phi , e , d ;

   Vec<ZZ> keys = KeyGen() ;
   n = keys[0] ;
   e = keys[1] ;
   d = keys[2] ;
   ZZ m , sigma ;

   m = 2567 ;
   ZZ h = Hash(m) ;         // Some hash of message M

   cout << "Message        : " << m << endl ;
   cout << "Hashed Message : " << h << endl ;
   cout << "----------------------------------------------------------------------
----------------" << endl ;

   sigma = Sign(m,d,n) ;
   cout << "Sign     : " << sigma << endl ;
   cout << "----------------------------------------------------------------------
----------------" << endl ;

   bool t = Verify(m,sigma,e,n) ;
   cout << "Pass 1" << endl ;
   cout << "Message : " << m << endl ;
   cout << "Sign     : " << sigma << endl ;
   cout << "Valid    : " << t << endl ;
   cout << "----------------------------------------------------------------------
----------------" << endl ;

   bool t1 = Verify(m,sigma + 1,e,n) ;
   cout << "Pass 2 - [With Tampering]" << endl ;
   cout << "Message : " << m << endl ;
   cout << "Sign     : " << sigma + 1 << endl ;
   cout << "Valid    : " << t1 << endl ;
   cout << "----------------------------------------------------------------------
----------------" << endl ;
}
```

Output :



```
➜  Submission git:(master) ✗ ./B180341CS_Prog4.out
Message        : 2567
Hashed Message : 1140304759920383833489745472737576249245587440647
--------------------------------------------------------------------------------------
Sign     : 4378108000919839158553246072301687441261135243712377707006366029582573508863302393989867407801864077880703714560439436592268142255013966701850848215559880
--------------------------------------------------------------------------------------
Pass 1
Message : 2567
Sign     : 4378108000919839158553246072301687441261135243712377707006366029582573508863302393989867407801864077880703714560439436592268142255013966701850848215559880
Valid    : 1
--------------------------------------------------------------------------------------
Pass 2 - [With Tampering]
Message : 2567
Sign     : 4378108000919839158553246072301687441261135243712377707006366029582573508863302393989867407801864077880703714560439436592268142255013966701850848215559881
Valid    : 0
--------------------------------------------------------------------------------------
```

Q4 - b : Digital Signature Implementation using ElGamal

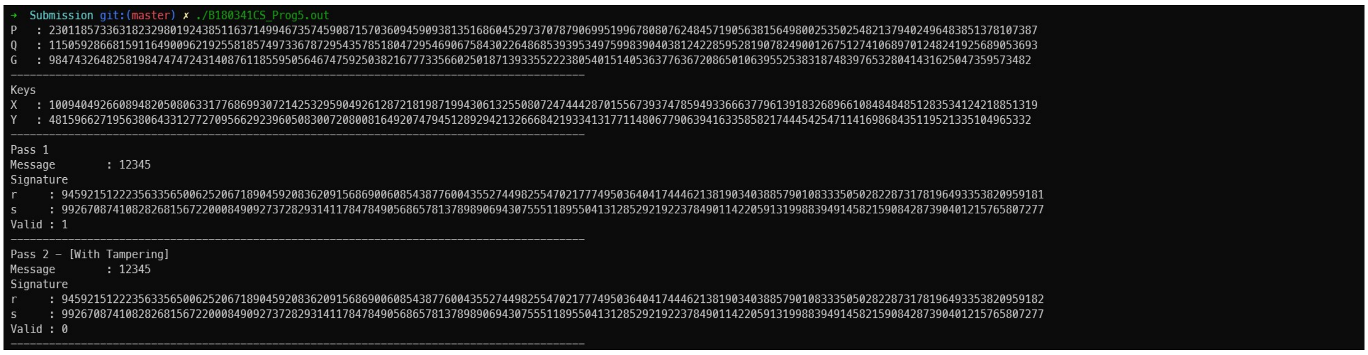
```
// Implementation of Digital Signature using ElGamal algorithm
// Hash related functions are stored in crypto.h

#include <NTL/ZZ.h>
#include <openssl/sha.h>
#include <bits/stdc++.h>
#include <string>
```

```
using namespace std;
using namespace NTL;

char *hexdigest(unsigned char *md, int len)
{
    static char buf[80];
    int i;
    for (i = 0; i < len; i++)
        sprintf(buf + i * 2, "%02x", md[i]);
    return buf;
}

ZZ hexToZZ(char *hex)
{
   ZZ res = ZZ(0);
   int i;
   for (i = 0; i < strlen(hex); i += 1)
   {
      res <<= 4;
      char x = hex[i];
      if(x>='0' && x <='9')
         res += hex[i]-48;
      else if(x>='a' && x <='f')
         res += hex[i]-87;
      else if(x>='A' && x <='F')
         res += hex[i]-55;
   }
   return res ;
}

string numberToString(ZZ num)
{
    string s = "";
    while (num > 0)
    {
        s += (num % 10) + '0';
        num /= 10;
    }
    reverse(s.begin(), s.end());
    return s;
}

// SHA1
ZZ Hash(string s)
{
    unsigned char hash[SHA_DIGEST_LENGTH]; // == 20

    SHA_CTX sha1;
    SHA1_Init(&sha1);
    SHA1_Update(&sha1, s.c_str(), s.length());
    SHA1_Final(hash, &sha1);

    ZZ h = hexToZZ(hexdigest(hash, SHA_DIGEST_LENGTH));
    return h ;
```

```
}

ZZ Hash(ZZ m)
{
    string s = numberToString(m);
    return Hash(s);
}

ZZ Hash(char *m)
{
    string s = m;
    return Hash(s);
}
#define BIT_LENGTH 512
#define ERR_THRESHOLD 1000

Vec<ZZ> Setup()
{
   Vec<ZZ> v ;
   v.FixLength(3) ;

   ZZ p , q , t , g , h;

   GenGermainPrime(q,BIT_LENGTH,ERR_THRESHOLD) ;
   p = 2*q + 1 ;

   g = 1 ;
   while(g==1)
   {
      h = RandomBnd(p) ;
      g = PowerMod(h,(p-1)/q,p) ;
   }

   v[0] = p ;
   v[1] = q ;
   v[2] = g ;
   return v ;
}

// Key Generation
Vec<ZZ> KeyGen(ZZ p,ZZ q,ZZ g)
{
   Vec<ZZ> v ;
   v.FixLength(2) ;

   ZZ x,y ;

   do {
      x = RandomBnd(q) ;
   } while (GCD(x,p)!=1) ;

   y = PowerMod(g,x,p);
   v[0] = x ;
   v[1] = y ;
   return v ;
}

// Sign
Vec<ZZ> Sign(ZZ m,ZZ p,ZZ q,ZZ g,ZZ x)
{
   Vec<ZZ> v ;
   v.FixLength(2) ;

   ZZ k,r,t ;

   do {
      k = RandomBnd(q) ;
      t = PowerMod(g,k,p) ;
      r = (t%q) ;
   } while (r==0) ;

   ZZ h = Hash(m) ;

   ZZ s = (InvMod(k,q)*(h+x*r))%q ;

   v[0] = r ;
   v[1] = s ;

   return v ;
}

// Verify
bool Verify(ZZ m,Vec<ZZ> v,ZZ y,ZZ p,ZZ q,ZZ g)
{
   ZZ r = v[0] ;
   ZZ s = v[1] ;

   if(r<0 || r>=q) return false ;
   if(s<0 || s>=q) return false ;

   ZZ h = Hash(m) ;

   ZZ w = InvMod(s,q) ;
   ZZ u1 = (h*w)%q ;
   ZZ u2 = (r*w)%q ;

   ZZ t = (PowerMod(g,u1,p) * PowerMod(y,u2,p))%p ;

   ZZ r_ = (t%q) ;

   if (r_==r)
      return true ;
   else
      return false ;
}

int main()
```

```
{
   ZZ p , q , n , phi , g , d ;

   Vec<ZZ> set = Setup() ;
   p = set[0] ;
   q = set[1] ;
   g = set[2] ;

   cout << "P   : " << p << endl ;
   cout << "Q   : " << q << endl ;
   cout << "G   : " << g << endl ;
   cout << "----------------------------------------------------------------------
--------------------" << endl ;

   ZZ x,y ;
   Vec<ZZ> keys = KeyGen(p,q,g) ;
   x = keys[0] ;
   y = keys[1] ;

   cout << "Keys" << endl ;
   cout << "X   : " << x << endl ;
   cout << "Y   : " << y << endl ;
   cout << "----------------------------------------------------------------------
--------------------" << endl ;

   ZZ m ;
   m = ZZ(12345) ;

   Vec<ZZ> sign = Sign(m,p,q,g,x) ;
   cout << "Pass 1" << endl ;
   cout << "Message        : " << m << endl ;
   cout << "Signature" << endl ;
   cout << "r     : " << sign[0] << endl ;
   cout << "s     : " << sign[1] << endl ;
   bool v = Verify(m,sign,y,p,q,g) ;
   cout << "Valid : " << v << endl ;
   cout << "----------------------------------------------------------------------
--------------------" << endl ;


   sign[0] = sign[0] + 1 ;
   cout << "Pass 2 - [With Tampering] " << endl ;
   cout << "Message        : " << m << endl ;
   cout << "Signature" << endl ;
   cout << "r     : " << sign[0] << endl ;
   cout << "s     : " << sign[1] << endl ;
   v = Verify(m,sign,y,p,q,g) ;
   cout << "Valid : " << v << endl ;
   cout << "----------------------------------------------------------------------
--------------------" << endl ;

   return 0 ;
}
```

Output :



```
➜  Submission git:(master) ✗ ./B180341CS_Prog5.out
P   : 2301185733631823298019243851163714994673574590871570360945909381351686045297370787906995199678080762484571905638156498002535025482137940249648385137810738 7
Q   : 115059286681591164900962192558185749733678729543578518047295469067584302264868539395349759983904038124228595281907824900126751274106897012482419256890536 93
G   : 9847432648258198474747243140876118559505646747592503821677733566025018713933552223805401514053637763672086501063955253831874839765328041431625047359573482
-------------------------------------------------------------------------------------------
Keys
X   : 10094049266089482050806331776869930721425329590492612872181987199430613255080724744428701556739374785949336663779613918326896610848484851283534124218851319
Y   : 4815966271956380643312772709566292396050830072080081649207479451289294213266684219334131771148067790639416335858217444542547114169868435119521335104965332
-------------------------------------------------------------------------------------------
Pass 1
Message        : 12345
Signature
r     : 9459215122235633565006252067189045920836209156869006085438776004355274498255470217774950364041744462138190340388579010833350502822873178196493353820959181
s     : 9926708741082826815672200084909273728293141178478490568657813789890694307555118955041312852921922378490114220591319988394914582159084287390401215765807277
Valid : 1
-------------------------------------------------------------------------------------------
Pass 2 - [With Tampering]
Message        : 12345
Signature
r     : 9459215122235633565006252067189045920836209156869006085438776004355274498255470217774950364041744462138190340388579010833350502822873178196493353820959182
s     : 9926708741082826815672200084909273728293141178478490568657813789890694307555118955041312852921922378490114220591319988394914582159084287390401215765807277
Valid : 0
-------------------------------------------------------------------------------------------
```

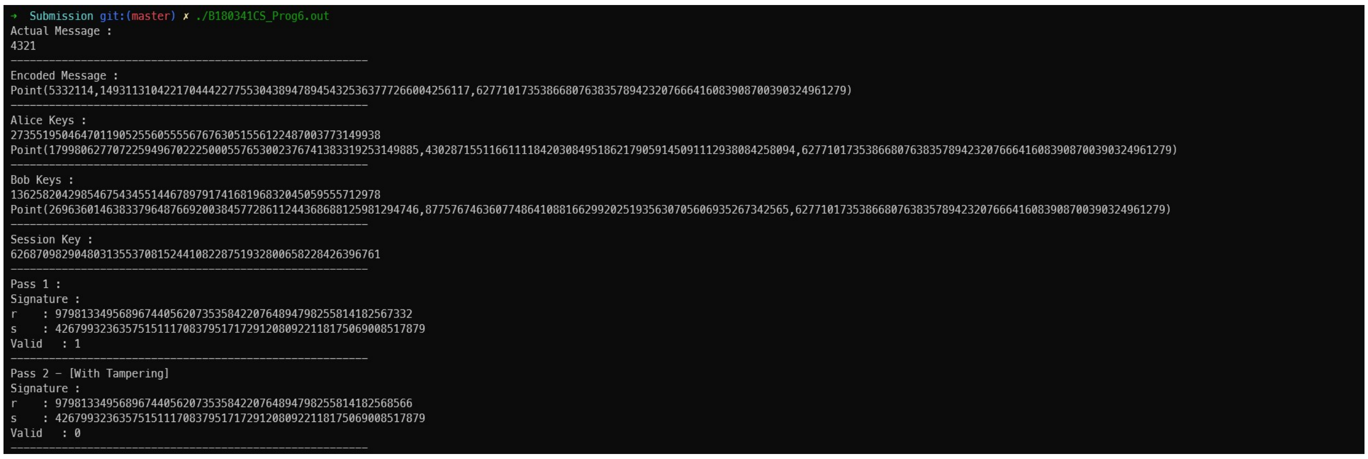
Q4 - c : Digital Signature Implementation using ECC

```
// Implementation of Digital Signature using ECC

#include <openssl/sha.h>
#include <bits/stdc++.h>
#include <NTL/ZZ.h>
#include <string.h>
#include <vector>
#include <string>
#include <bitset>

using namespace std;
using namespace NTL;

ZZ modInverse(ZZ x,ZZ p)
{
    for(ZZ i=ZZ(0);i<p;i+=1)
        if(MulMod(i,x,p) == 1)
            return i ;
    return ZZ(-1) ;
}

class Point {
    private:
        ZZ p ;
        ZZ a ;
        ZZ b ;
        string zztostring(ZZ num)
        {
            long len = ceil(log(num)/log(128));
            char str[len];
            for(long i = len-1; i >= 0; i--)
            {
                str[i] = conv<int>(num % 128);
                num /= 128;
            }

            return (string) str;
        }
    public:
        ZZ x ;
        ZZ y ;

        Point(ZZ x_,ZZ y_,ZZ p_,ZZ a_=ZZ(0),ZZ b_=ZZ(0))
        {
            x = x_ ;
            y = y_ ;
            p = p_ ;
            a = a_ ;
            b = b_ ;
        }

        Point(const Point &P)
        {
            x = P.x ;
            y = P.y ;
            p = P.p ;
            a = P.a ;
            b = P.b ;
        }

        ZZ get_a()
        {
            return a ;
        }

        ZZ get_b()
        {
            return b ;
        }

        ZZ get_p()
        {
            return p ;
        }

        void display()
        {
            cout << "Point("
                 << x << ","
                 << y << ","
                 << p << ")" << endl ;
        }

        string toString()
        {
            string str = "Point(" ;
            str += zztostring(x) ;
            str += "," ;
            str += zztostring(y) ;
            str += "," ;
            str += zztostring(p) ;
            str += ")" ;
            return str ;
        }

        Point inverse()
        {
            Point P = copy() ;
            P.y = (-P.y) % P.p ;
            return P ;
        }

        Point copy()
        {
            Point P = Point(x,y,p,a,b) ;
            return P ;
        }

        bool Eq(Point const &obj)
        {
            if(obj.x == x && obj.y == y && obj.p == p && obj.a == a && obj.b == b)
                return 1 ;
            else
                return 0 ;
        }

        Point Double()
        {
            ZZ l ;
            ZZ t_y ;
            try {
                t_y = InvMod(MulMod(ZZ(2),y,p),p) ;
            } catch(InvModErrorObject &e) {
                cout << "Error: " << e.what() << endl ;
                return Point(ZZ(-1),ZZ(-1),ZZ(-1)) ;
            }
            catch(...) {
                cout << "[!] Double : Non Invertible Point" << endl ;
                return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
            }
            l = MulMod(ZZ(3) * power(x,2) + a,t_y,p) ;

            ZZ fx = (power(l,2)-x-x) % p ;

            ZZ t = x - fx ;
            ZZ fy = (l*t - y) % p ;

            return Point(fx,fy,p,a,b) ;
        }

        ostream& operator<<(ostream &out)
        {
            out << "Point(" << x << "," << y << "," << p << "," << a << "," << b
<< ")" ;
            return out ;
        }
        Point operator + (Point const &obj)
        {
            if(p != obj.p)
                cout << "[+] Invalid Operands" << endl ;
            ZZ y_ = obj.y - y ;
            ZZ x_ = obj.x - x ;

            if(obj.y == 0 && obj.x == 0)
                return Point(x,y,p,a,b) ;
            if(y == 0 && x == 0)
                return Point(obj.x,obj.y,obj.p,obj.a,obj.b) ;

            ZZ l ;
            if(Eq(obj))
            {
                ZZ t_y ;
                try {
                    t_y = InvMod(MulMod(ZZ(2),y,p),p) ;
                } catch(...) {
                    cout << "[!] + if : Non Invertible Point" << endl ;
                    return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
                }
                l = MulMod(ZZ(3) * power(x,2) + a,t_y,p) ;
            }
            else
            {
                ZZ t_x ;
                try {
                    x_ = x_ % p ;
                    t_x = InvMod(x_,p) ;
                } catch(...) {
                    cout << "[!] + else : Non Invertible Point : " << x_ << endl ;
                    cout << "[!] GCD : " << GCD(x_,p) << endl ;
                    return Point(ZZ(-1),ZZ(-1),ZZ(p),ZZ(a),ZZ(b)) ;
                }
                l = MulMod(y_,t_x,p) ;
            }
            ZZ fx = (power(l,2)-x-obj.x) % p ;
            ZZ t = x - fx ;
            ZZ fy = (l*t - y) % p ;

            return Point(fx,fy,p,a,b) ;
        }

        Point operator * (ZZ const &obj)
        {
            return scalarMul(obj) ;
        }

        Point operator * (int const &obj)
        {
            return scalarMul(ZZ(obj)) ;
        }
        // Scalar Multiplication of Point
        Point scalarMul(ZZ const &k)
        {
            Point P = copy() ;
            Point ans = Point(ZZ(0),ZZ(0),p,a,b) ;
            unsigned char* p = new unsigned char[NumBytes(k)];
            BytesFromZZ(p, k, NumBytes(k)); // pp = byte-representation of N
            for(int i=0;i<NumBytes(k);i+=1)
            {
                bitset<8> x(p[i]) ;         // x = binary representation of p[i]
                for(int j=0;j<8;j+=1)
                {
                    if(x[j] == 1)
                        ans = ans + P ;
                    P = P.Double() ;
                }
            }
            delete[] p;
            return ans ;
        }
} ;

char *hexdigest(unsigned char *md, int len)
{
    static char buf[80];
    int i;
    for (i = 0; i < len; i++)
        sprintf(buf + i * 2, "%02x", md[i]);
    return buf;
}

ZZ hexToZZ(char *hex)
{
   ZZ res = ZZ(0);
   int i;
   for (i = 0; i < strlen(hex); i += 1)
   {
      res <<= 4;
      char x = hex[i];
      if(x>='0' && x <='9')
         res += hex[i]-48;
      else if(x>='a' && x <='f')
         res += hex[i]-87;
      else if(x>='A' && x <='F')
         res += hex[i]-55;
   }
   return res ;
}

string numberToString(ZZ num)
{
    string s = "";
    while (num > 0)
    {
        s += (num % 10) + '0';
        num /= 10;
    }
    reverse(s.begin(), s.end());
    return s;
}

// SHA1
ZZ Hash(string s)
{
    unsigned char hash[SHA_DIGEST_LENGTH]; // == 20

    SHA_CTX sha1;
    SHA1_Init(&sha1);
    SHA1_Update(&sha1, s.c_str(), s.length());
    SHA1_Final(hash, &sha1);

    ZZ h = hexToZZ(hexdigest(hash, SHA_DIGEST_LENGTH));
    return h ;
}

ZZ Hash(ZZ m)
{
    string s = numberToString(m);
    return Hash(s);
}

ZZ Hash(char *m)
{
    string s = m;
    return Hash(s);
}

#define K_VALUE 1234

typedef struct Key
{
   ZZ privateKey ;
   Point publicKey ;
} Key ;

Key KeyGen(ZZ n,Point B)
{
   ZZ PrivateKey = RandomBnd(n) ;
   Point PublicKey = B * PrivateKey ;
   Key k = {PrivateKey,PublicKey} ;
   return k ;
}

Point messageEncode(ZZ m,Point BasePoint)
{
   ZZ k = ZZ(K_VALUE) ;
   ZZ Xj , Yj , Sj ;
```

```
    for(int j=0;j<k;j+=1)
    {
       Xj = k * m + j ;
       Sj = power(Xj,3) + BasePoint.get_a()*Xj + BasePoint.get_b() ;
       Sj = Sj % BasePoint.get_p() ;
       if(Jacobi(Sj,BasePoint.get_p())==1)
       {
          Yj = PowerMod(Sj,(BasePoint.get_p()+1)/ZZ(4),BasePoint.get_p()) ;
          return Point(Xj,Yj,BasePoint.get_p(),BasePoint.get_a(),BasePoint.get_b())
;
       }
    }
    return BasePoint * m ;
}

ZZ messageDecode(Point Pm,Point BasePoint)
{
    ZZ k = ZZ(K_VALUE) ;
    return Pm.x / k ;
}

// Sign
Vec<ZZ> Sign(Point Pm,ZZ privateKey,ZZ sessionKey,ZZ n,Point BasePoint)
{
    Vec<ZZ> v ;
    v.FixLength(2) ;

    Point R = BasePoint * sessionKey ;
    ZZ r = R.x ;

    ZZ s = (InvMod(sessionKey,n) * (Hash(Pm.toString()) + privateKey * r)) % n ;

    v[0] = r ;
    v[1] = s ;

    return v ;
}

// Verify
bool Verify(Vec<ZZ> key,ZZ n,Point Pm,Point publicKey,Point BasePoint)
{
    ZZ r = key[0] ;
    ZZ s = key[1] ;

    ZZ w = InvMod(s,n) ;

    ZZ u = (Hash(Pm.toString()) * w)%n ;
    ZZ v = (r * w)%n ;

    Point R = (BasePoint * u) + (publicKey * v) ;

    if(R.x==r)
       return true ;
    else
        return false ;
}

int main()
{
    Vec<ZZ> curveParams ;
    curveParams.SetLength(3) ;
    curveParams[0] = power(ZZ(2),192) - power(ZZ(2),64) - ZZ(1) ;   // p
    curveParams[1] = ZZ(-3) ;                                       // a
    curveParams[2] = conv<ZZ>
("2455155546008943817740293915197451784769108058161191238065") ;    // b

    ZZ p = curveParams[0] ;
    ZZ a = curveParams[1] ;
    ZZ b = curveParams[2] ;

    ZZ n = conv<ZZ>("6277101735386680763835789423176059013767194773182842284081") ;
// n - Order
    ZZ seed = conv<ZZ>("275585503993527016686210752207080241786546919125") ;
    ZZ h = ZZ(1) ;
    ZZ r = conv<ZZ>("1191689908718309326471930603292001425137626342642504031845") ;

    ZZ x = conv<ZZ>("602046282375688656758213480587526111916698976636884684818") ;
    ZZ y = conv<ZZ>("174050332293622031404857552280219410364023488927386650641") ;

    Point BasePoint = Point(
    x,              // x
    y,              // y
    curveParams[0],
    curveParams[1],
    curveParams[2]
    ) ;

    Key key = KeyGen(n,BasePoint) ;

    int message = 4321 ;
    ZZ m = ZZ(message) ;

    Point Pm = messageEncode(m,BasePoint) ;

    cout << "Actual Message : " << endl ;
    cout << m << endl ;
    cout << "--------------------------------------------------------" << endl ;

    cout << "Encoded Message : " << endl ;
    Pm.display() ;
    cout << "--------------------------------------------------------" << endl ;

    Key aliceKey = KeyGen(n,BasePoint) ;
    cout << "Alice Keys : " << endl ;
    cout << aliceKey.privateKey << endl ;
    aliceKey.publicKey.display() ;
    cout << "--------------------------------------------------------" << endl ;
```

```cpp
    Key bobKey = KeyGen(n,BasePoint) ;
    cout << "Bob Keys : " << endl ;
    cout << bobKey.privateKey << endl ;
    bobKey.publicKey.display() ;
    cout << "--------------------------------------------------------" << endl ;

    ZZ k ;
    do {
        k = RandomBnd(n) ;
    } while (GCD(k,n)!=1) ;

    cout << "Session Key : " << endl ;
    cout << k << endl ;
    cout << "--------------------------------------------------------" << endl ;

    Vec<ZZ> sign = Sign(Pm,bobKey.privateKey,k,n,BasePoint) ;
    cout << "Pass 1 : " << endl ;
    cout << "Signature : " << endl ;
    cout << "r     : " << sign[0] << endl ;
    cout << "s     : " << sign[1] << endl ;
    bool t = Verify(sign,n,Pm,bobKey.publicKey,BasePoint) ;
    cout << "Valid   : " << t << endl ;
    cout << "--------------------------------------------------------" << endl ;

    sign[0] = sign[0] + 1234 ;
    cout << "Pass 2 - [With Tampering] " << endl ;
    cout << "Signature : " << endl ;
    cout << "r     : " << sign[0] << endl ;
    cout << "s     : " << sign[1] << endl ;
    t = Verify(sign,n,Pm,bobKey.publicKey,BasePoint) ;
    cout << "Valid   : " << t << endl ;
    cout << "--------------------------------------------------------" << endl ;
}
```

Output :



```
→  Submission git:(master) ✗ ./B180341CS_Prog6.out
Actual Message :
4321
--------------------------------------------------------
Encoded Message :
Point(5332114,1493113104221704442277553043894789454325363777266004256117,6277101735386680763835789423207666416083908700390324961279)
--------------------------------------------------------
Alice Keys :
2735519504647011905255605555676763051556122487003773149938
Point(1799806277072259496702225000557653002376741383319253149885,4302871551166111184203084951862179059145091112938084258094,6277101735386680763835789423207666416083908700390324961279)
--------------------------------------------------------
Bob Keys :
1362582042985467543455144678979174168196832045059555712978
Point(2696360146383379648766920038457728611244368688125981294746,8775767463607748641088166299202519356307056069352673425 65,6277101735386680763835789423207666416083908700390324961279)
--------------------------------------------------------
Session Key :
6268709829048031355370815244108228751932800658228426396761
--------------------------------------------------------
Pass 1 :
Signature :
r     : 9798133495689674405620735358422076489479825581418256733 2
s     : 4267993236357515111708379517172912080922118175069008517879
Valid   : 1
--------------------------------------------------------
Pass 2 - [With Tampering]
Signature :
r     : 9798133495689674405620735358422076489479825581418256856 6
s     : 4267993236357515111708379517172912080922118175069008517879
Valid   : 0
--------------------------------------------------------
```